# Evolutionary Algorithms – Final Project

## Gal "Hanochel" Dahan – 322818014, Itay "GotABigOne" Tabib – 206577975

# Longest Simple Path Problem with Genetic Algorithms

## A short Introduction

LSP (longest simple path) problem is a NP-hard known problem in graph theory. In this problem, we aim to find the longest path in a graph, where no node is visited more than once.

For this project, we will focus on LSP in generic undirected graphs. Because this problem is NP-hard, solutions for this problem is mostly based on heuristics.

There are a lot of domains and motivation to solve this problem like information retrieval on peer-to-peer networks, estimating the worst packet delay of Switched Ethernet network, or ﬁnding long paths in topological graph-like maps, which usually do not contain Euler or Hamilton paths, to use as a reference for patrolling an environment with multiple robots.

In this project, we will show methods for solving LSP problem, using heuristic search and compare them to a genetic algorithm-based solution.

# Solving LSP with heuristic search:

# Solving LSP with genetic algorithms

Genetic algorithms (GAs) have become very popular for solving complex optimization and search problems. As a consequence, applications on graph theory problems have emerged. GAs can collect and understand important knowledge about the search space during the search space, and adjust future iterations based on the information previously obtained through stochastic optimization techniques. Taking into account the inherent knowledge in the search space, it is more likely to obtain the global optimal solution using genetic algorithms instead of traditional search algorithms.

# Algorithms

We will take a look at several genetic algorithms for LSP and compare them to the heuristic search-based solutions.

## Representation

Let be a connected undirected weighted graph with no self-edges.

All the genetic algorithms will have the same representation for the solution space which will be a list of numbers representing vertices in the graph that the list of vertices is a legal path in the graph.

In other words a solution will be represented by a list that .

## Initialization

Generating initial solutions with good quality for this problem is very important because the algorithms strongly rely on the genetic material of the initial solution space. Initially, a random method was considered. It started by selecting a random vertex of the graph and compute paths by choosing neighbors of the current vertex at random, as long as they were not already included in the path. It ﬁnished computing the path when there were no available neighbors left. This method was inefficient because the algorithm chose vertices with only one neighbor and the initial paths were very short.

To overcome this fact, a new method was proposed:

The ﬁrst vertex is still selected at random, however, the following vertices are selected with a probability according to their degree. In addition, if we have reached a vertex with unavailable neighbors, instead of stopping the method, we analyze whether the degree of the ﬁrst vertex is greater than one and keep computing the path to the opposite direction until reaching a ﬁnishing point for the method.

This technique presented much better results for initialization good first population.

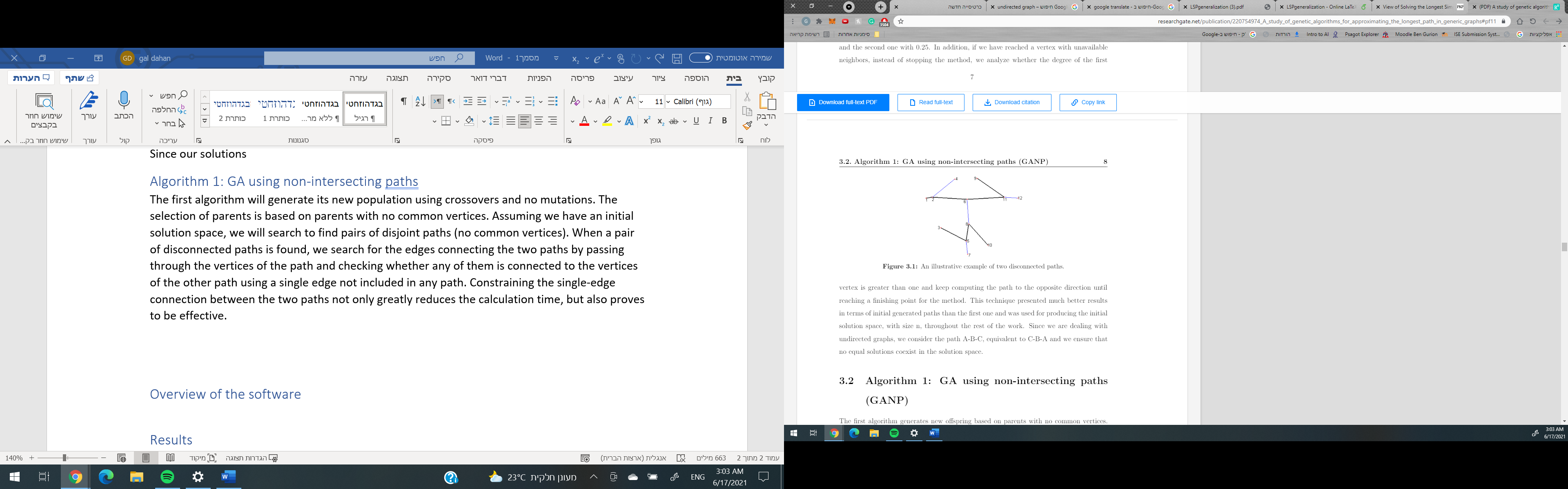
Since we are dealing with undirected graphs, we consider the path A-B-C, equivalent to C-B-A and we ensure that no equal solutions coexist in the solution space.

## Fitness

Our fitness function will be for each solution, compute its path cost.

## Algorithm 1: GA using non-intersecting paths

The first algorithm will generate its new population using crossovers and no mutations. The selection of parents is based on parents with no common vertices. Assuming we have an initial solution space, we will search to find pairs of disjoint paths (no common vertices). When a pair of disconnected paths is found, we search for the edges connecting the two paths by passing through the vertices of the path and checking whether any of them is connected to the vertices of the other path using a single edge not included in any path. Constraining the single-edge connection between the two paths not only greatly reduces the calculation time, but also proves to be effective.



For example, we can see and are two disconnected paths and the edge is connecting the paths. As a result, we generate 4 offspring:

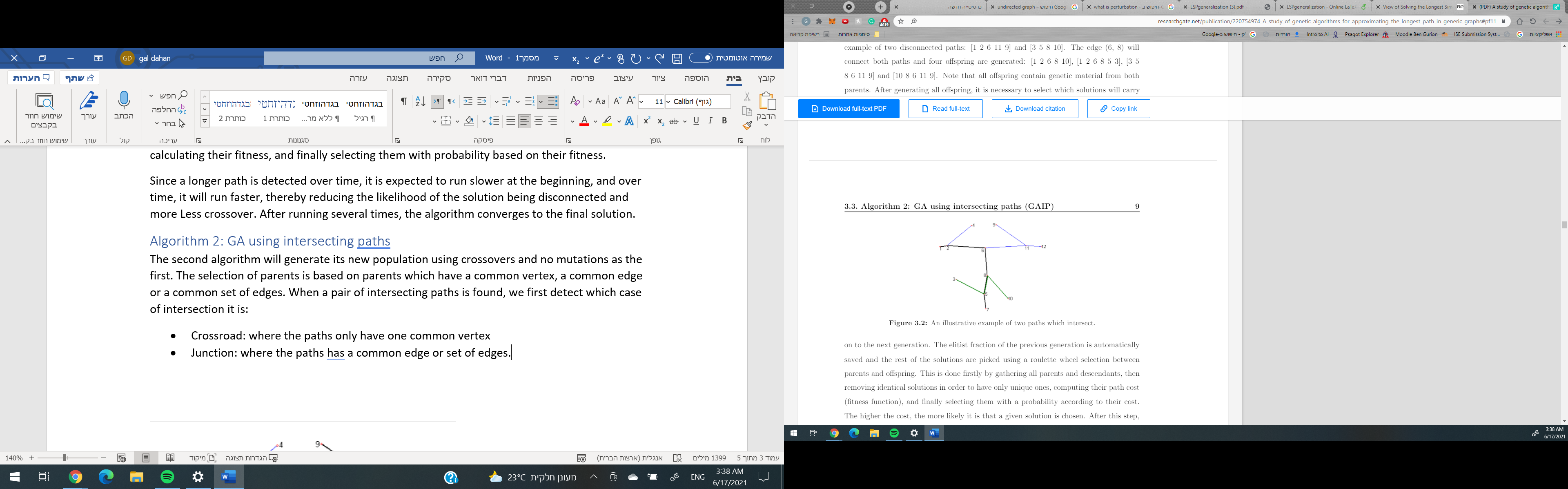
After generating all the offspring, we need to select which solutions will carry on to the next generation. The elite part of the previous generation is automatically saved, and the rest of the solution is selected using the roulette selection between parents and offspring. This is first done by collecting all parents and offspring, then deleting the same solutions to get unique solutions, calculating their fitness, and finally selecting them with probability based on their fitness.

Since a longer path is detected over time, it is expected to run slower at the beginning, and over time, it will run faster, thereby reducing the likelihood of the solution being disconnected and less crossovers are made. After running several times, the algorithm converges to the final solution.

## Algorithm 2: GA using intersecting paths

The second algorithm will generate its new population using crossovers and no mutations as the first. The selection of parents is based on parents which have a common vertex, a common edge, or a common set of edges. When a pair of intersecting paths is found, we first detect which case of intersection it is:

* Crossroad: where the paths only have one common vertex
* Junction: where the paths have a common edge or set of edges.



For example, we can see and are two intersected paths that have a common edge As a result we generate 2 offspring: . It is worth mentioning that if the junction occurs in the beginning (or end) of one or both paths, no oﬀspring is generated.

After generating all the offspring, we act as the first algorithm.

Since a longer path is detected over time, it is expected to run faster at the beginning, and over time, it will run slower, thereby increasing the likelihood of the solution being connected and more crossovers are made.

## Algorithm 3: GA using intersecting and non-intersecting paths

The third algorithm basically combines the two algorithms shown. The selection of parents is now based on paths that intersect once and paths that are disconnected. Offspring will be generated as seen for each case. After generating all oﬀspring, the selection of paths to carry on to the next generation and the stopping condition are the same as in the previous algorithms.

Since a longer path is detected over time, it is expected to run slower at the beginning, and over time, it will run faster, thereby reducing the likelihood of the solution being disconnected and more Less crossover. After running several times, the algorithm converges to the final solution.

## Overview of the software

idk

## Results

## Conclusion

## Bibliographic references

* Yossi Cohen, Roni Stern, and Ariel Felner. Solving the longest simple path problem with heuristic search. Proceedings of the International Conference on Automated Planning and Scheduling, 30(1):75–79, Jun. 2020.
* Giovanni Giardini, Tamás Kalmár-Nagy, "Genetic Algorithm for Combinatorial Path Planning: The Subtour Problem", Mathematical Problems in Engineering, vol. 2011, Article ID 483643, 31 pages, 2011. https://doi.org/10.1155/2011/483643
* Portugal, D.; Antunes, C. H.; and Rocha, R. P. 2010. A study of genetic algorithms for approximating the longest path in generic graphs. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10-13 October 2010, 2539–2544.